(2)

AD-A197 319

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THREE DIMENSIONAL VISUAL DISPLAY
FOR A PROTOTYPE
COMMAND AND CONTROL WORKSTATION

by

Milton D. Abner

June 1988

Thesis Advisor: Michael J. Zyda

Approved for public release; distribution is unlimited

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL *(If applicable)* Code 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000		7b ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL *(If applicable)*	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS *(City, State, and ZIP Code)*		10 SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

11 TITLE *(Include Security Classification)*
Three Dimensional Visual Display for a Prototype Command and Control Workstation

12 PERSONAL AUTHOR(S)
Abner, Milton D.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT *(Year, Month, Day)*	15 PAGE COUNT
Master's Thesis	FROM ________ TO ________	1988 June	68

16 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	Real-time; Shading model; Lighting model

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The development of a real-time three-dimensional visual display for the Command and Control Workstation of the Future (CCWF) is a means of rapidly interpreting large amounts of important information. In this study, we examine the realistic versus real-time trade-offs required to achieve such a display and the components effecting these trade-offs, i.e., hidden surface technique, lighting and shading models, etc. We also present a unified data structure that is used in storing various properties that create the display.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Michael J. Zyda	22b TELEPHONE *(Include Area Code)* (408) 646-2305	22c OFFICE SYMBOL Code 52Zk

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24.

THREE DIMENSIONAL VISUAL DISPLAY FOR A PROTOTYPE COMMAND AND CONTROL WORKSTATION

by

Milton D. Abner
Lieutenant, United States Navy
B.S., Grambling State University, 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1988

Author: Milton D. Abner
Milton D. Abner

Approved by: [illegible]
Michael J. Zyda, Thesis Advisor

[illegible]
C. Thomas Wu, Second Reader

[illegible]
for Robert B. McGhee, Acting Chairman
Department of Computer Science

[illegible]
James M. Fremgen, Acting
Dean of Information and Policy Sciences

ABSTRACT

The development of a real-time three-dimensional visual display for the Command and Control Workstation of the Future (CCWF) is a means of rapidly interpreting large amounts of important information. In this study, we examine the realistic versus real-time trade-offs required to achieve such a display and the components effecting these trade-offs, i.e., hidden surface technique, lighting and shading models, etc. We also present a unified data structure that is used in storing various properties that create the display.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. THREE-DIMENSIONAL VISUAL DISPLAY FOR A PROTOTYPE COMMAND AND CONTROL WORKSTATION

Since the last global conflict, there has been a dramatic change in the way the armed forces are equipped to fight. The advent of long range airborne early warning radar systems, supersonic aircraft, and computer data networks has both expanded the commander's horizon and attempted to give him the ability to keep track of events that directly affect his environment. As one can readily see, today's commander has an abundance of information at his disposal. However, a major concern that exists is the amount of time required to interpret this information. All too often, large amounts of information are considered useless simply because of the way the information is presented [Ref.1].

With ample time, all the information provided by various sensors can be interpreted by today's commander. However, in cases where decisions have to be made in a matter of seconds, the commander needs to be able to view and understand incoming data on a real-time basis. It is this need that has prompted our work on a three-dimensional computer graphics addition to the Command and Control Workstation of the Future (CCWF).

In making rapid, sound decisions, the commander must have situational awareness better known as the big picture of what is going on around him. One of quickest ways to obtain situational awareness is to apply an old saying, "one picture is worth a thousand words". The CCWF allows the commander to access and interpret data rapidly by providing a three-dimensional situational view based on inputs from various sensors.

When referring to a situational view, we are talking about such displays as a view from the bridge of a ship, where other ships, aircraft, submarine periscopes, and hazards can be seen. In such a system, a major problem is the development of realistic three-dimensional visual images in real-time. When referring to computer graphics, the term real-time implies that complex pictures can be generated so rapidly that a display can be refreshed at a rate fast enough to appear continuous [Ref. 1]. The objective of this study is the development of visualization tools and techniques to aid in the design and implementation of three-dimensional visual displays for use in the CCWF.

1. Discussion

When developing realistic three-dimensional visual displays, many factors are involved. These factors range from the capabilities of the hardware utilized, to the various software techniques to be used. During this study, many questions on proposed techniques were asked, such as: which hidden surface removal technique should be used and what lighting and shading techniques should be used. These questions were asked because their solutions directly affect our ability to generate our displays in real-time.

B. METHODOLOGY

The computer graphics workstation on which this study was conducted is the IRIS-4D/70G. To achieve both an image which appears more realistic and one that can be displayed in real-time or near real-time requires some trade offs. This study begins by determining a list of variables that affect both the system's real-time capability and the realism of the images displayed. This list consists of such things as hidden surface removal techniques, lighting and shading techniques, and numbers of polygons. These variables as well as others are discussed in detail in later chapters. After establishing the list, a comparative study is conducted on the techniques to determine which is the best one for the specific application.

C. ORGANIZATION

The remainder of this study is organized in the following manner. Chapter II takes a look at the IRIS-4D/70G's lighting and shading capabilities. Chapter III describes different hidden surface removal techniques that were considered and some advantages and disadvantages of each. Chapter IV discusses the actual implementation and problems encountered during the implementation. Chapter V provides the conclusion reached from the project and discusses opportunities for future research.

II. LIGHTING AND SHADING CAPABILITIES OF THE IRIS-4D/70G

As we discuss the lighting capabilities of the IRIS-4D, we will try to obtain an understanding of the lighting models and what data is required to execute the calculations. In order to accomplish this, we must take a close look at the lighting equation used. As stated in [Ref.2:pp. 14.4] , the color of a point at position P_p is C_p and is the emitted light plus the sum of the ambient, diffuse, and specular light reflected by the point towards the eye. The equation for C_p is as follows:

$$C_p = C_{emitted} + C_{ambient} + C_{diffuse} + C_{specular}$$

A. EMITTED TERM

The emitted term, $C_{emitted}$, models the light emitted from self luminous material. Emitted light is independent of everything except the material's emission color and its intensity, therefore

$$C_{emitted} = C_{me}$$

where

- C_{me} is the emission color of the material.

The emission color of the material is represented by RGB colors where red, green, and blue are scalar values that range from 0.0 to 1.0, where 1.0 is the maximum intensity of a color. For the purpose of this study, all color specifications are in the range of 0.0 to 1.0.

B. AMBIENT TERM

The ambient term, $C_{ambient}$, models the intensity of reflection from a point on the surface of an object due to the ambient light source. There are two sources of ambient light, one being light which comes from the scene itself and the other being light which comes from a light source. The data necessary to compute the ambient term is summarized in the following equation:

$$C_{ambient} = C_{sa} C_{ma} + C_{la} C_{ma}$$

where

- C_{sa} is the color of the ambient light in the scene.
- C_{ma} is the ambient color of the material.
- C_{la} is the ambient color of the light.

C. DIFFUSE TERM

The diffuse term, $C_{diffuse}$, models the intensity contribution from diffuse reflection of incident light from a point source. Lambert's cosine law which states that an intensity of light reflected from a perfect diffuser is proportional to the cosine of the angle between the light direction and the normal to the surface [Ref.3], is used to compute the intensity of the diffuse light. The following equation summarizes the data required to compute the diffuse term.

$$C_{diffuse} = C_l C_{md} [N_{Pl} N_P]$$

where

- C_l is the color of the light.
- C_{md} is the diffuse color of the material.
- N_{Pl} is the direction from point P_P to the light.
- N_P is the normal to the surface at point P_P .

D. SPECULAR TERM

The specular term, $C_{specular}$, models the intensity of specularly reflected light. The intensity of the specular light is dependent on the angle between the surface normal ($\vec{N}_P$) and the bisector ($\vec{N}_b$) of the point to eye vector ($\vec{N}_{Pe}$) and the point to light vector ($\vec{N}_{Pl}$). As the angle between the surface normal and the bisector decreases, the intensity increases until the angle between the two vectors is zero, at which time the maximum intensity is established as illustrated in Figure 2.1. The data necessary to compute the specular term is summarized in the following equation:

$$C_{specular} = C_l C_{ms} [N_b N_P]^{Emss}$$

where

- C_l is the color of the light.
- N_P is the normal to the surface at point P_P .
- C_{ms} is the specular color of the material.
- E_{mss} is the material's specular scattering exponent.
- N_b is the bisector angle.

The variable E_{mss} is used to determine the angular range for viewing the specular reflection. If E_{mss} is small, then the angular range for viewing specular reflection is large as indicated in Figure 2.2. and the surface appears dull. A small E_{mss} value indicates a dull surface. If the value of E_{mss} is large, then the angular range for viewing specular reflection is small, indicating a shiny surface as indicated in Figure 2.3.

E. SHADING MODELS

When light strikes a surface, one of three possibilities can occur: the light can be absorbed, it can be reflected, or it can be transmitted. Some of the light is absorbed and converted into heat. Some of it passes through the surface, transmitted light, allowing

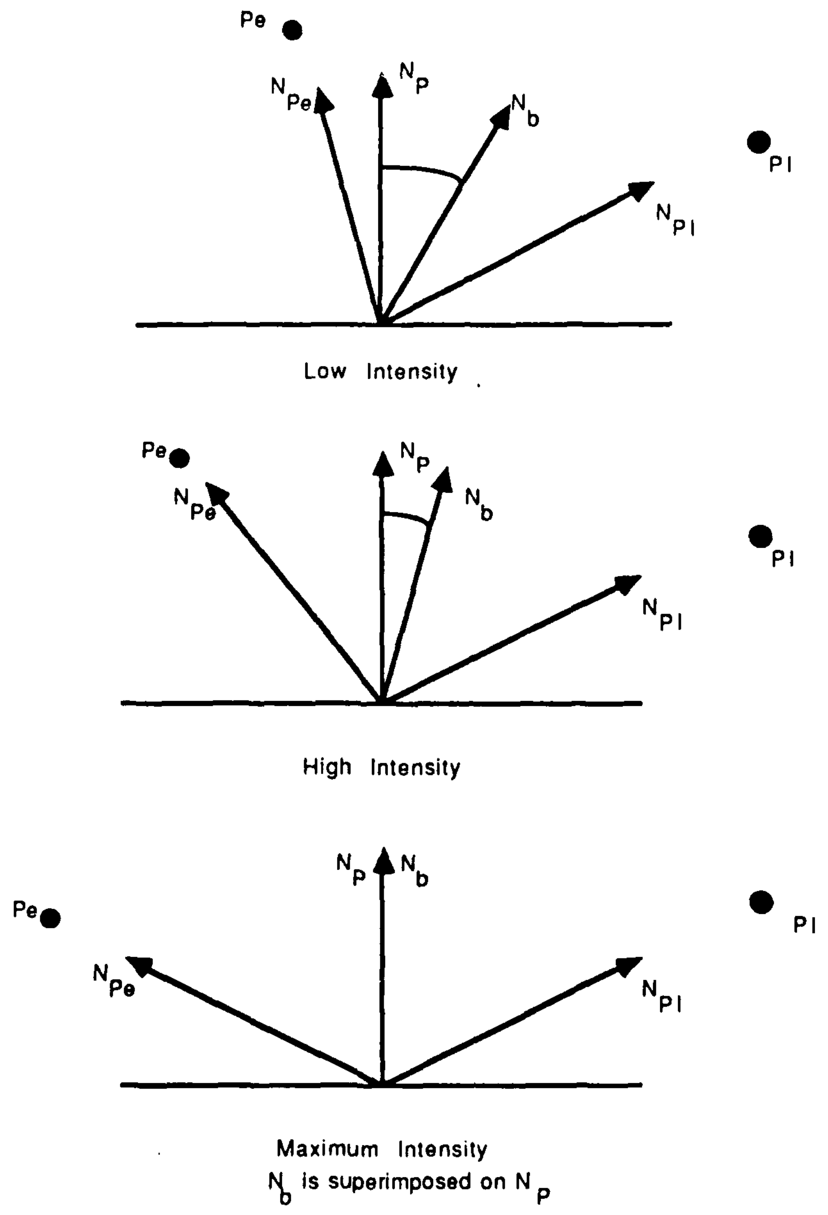


Figure 2.1 Varying Intensity of Specular Reflection

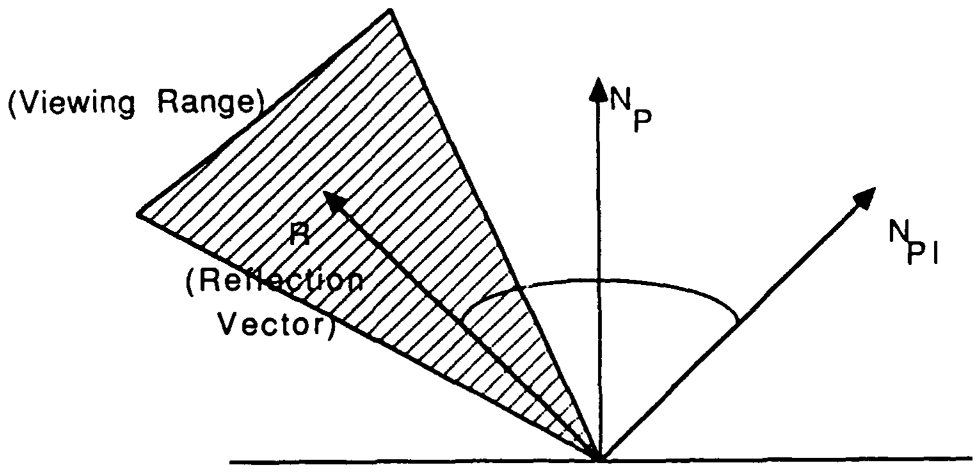


Figure 2.2 Small E_{mss}/ Dull Surface

the surface to have a transparent quality. However, it is the light that is reflected by the surface that allows an object to be visible.

A shading model is used to calculate the intensity of this reflected light and perform the color assignment that we should see when viewing a surface. In actuality, it is the illumination model within the shading model that calculates the the intensity of the reflected light. These intensity calculations are based on the optical properties of the surface, the relative positions of the surface, and their orientation with respect to light source [Ref.3:pp. 276].

There are several shading model methods such as: constant intensity, Gouraud shading, Phong shading, just to name a few. However, there are only two such techniques that are implemented on the IRIS-4D/70G that allow the intensity calculations

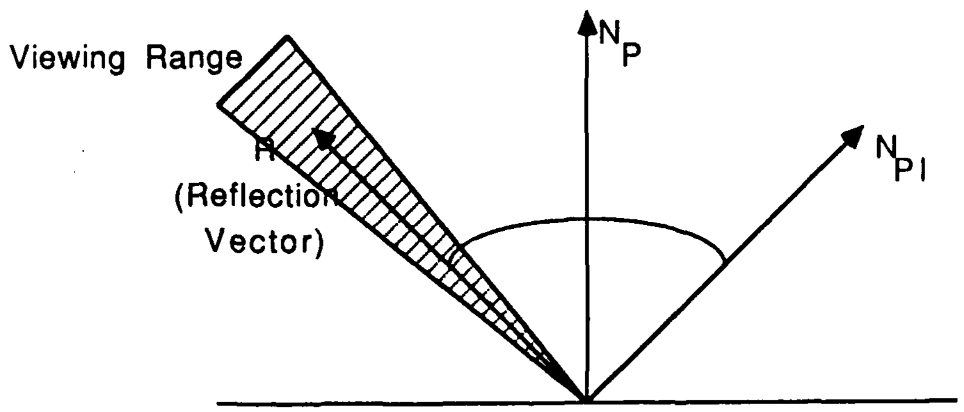


Figure 2.3 Large E_{mss} / Shiny Surface

to be computed in real-time. Those two techniques are constant intensity and Gouraud shading.

1. Constant Intensity

This is a very simple technique. When determining the intensity of the reflected light, one needs only to compute the intensity of the polygonal surface. This is accomplished by the illumination model which uses a surface normal and the light vector. Once we compute the intensity of the polygon, we know the color of every pixel in that polygon because they are all the same.

a. Performance

The constant intensity method allows for extremely rapid execution time. However there are two major problems with this method. One problem is, it is very difficult to generate an accurate representation of any type of surface other than a plane surface. A curved surface that is represented as a set of plane surfaces can be shaded with constant surface intensities if the planes subdividing the surface are made small enough [Ref.3:pp. 289]. However, this creates another question; just how small is small enough? As the plane surfaces becomes smaller, they began to increase in number, which starts to consume more processing time. The second problem that exists is, when the orientation between adjacent planes changes abruptly, as one might see when constructing the outer surface of a cylinder while using a relative few number of polygons, the difference in surface intensity can produce a harsh and unrealistic effect [Ref.3:pp.289]. Although there exist problems with constant intensity, there are areas where this method can generate accurate representation.

2. Gouraud Shading

Gouraud's method uses an *intensity interpolation* scheme which removes intensity discontinuities between adjacent planes of a surface representation [Ref.3:pp. 289]. In determining the intensity of reflected light from a surface, this

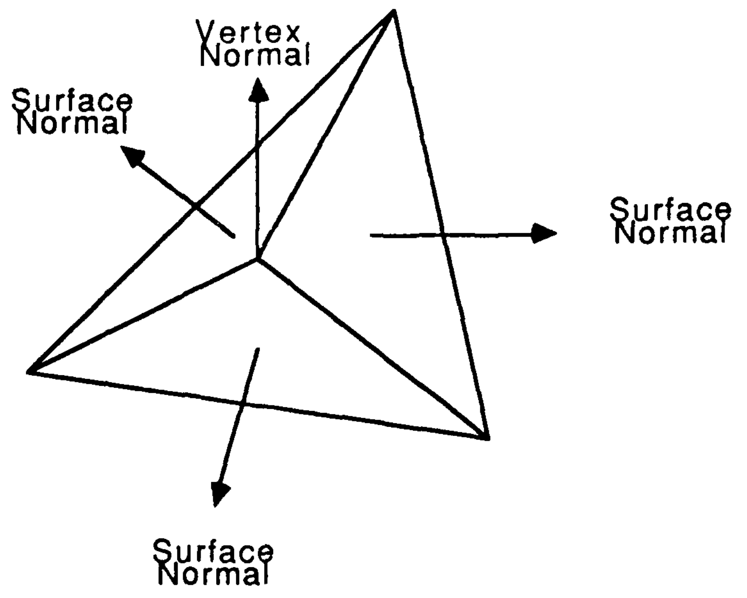


Figure 2.4 Vertex Normal

technique requires a great deal more computation than the constant intensity technique. Gouraud's technique utilizes a scan line algorithm to render the object. A value for the intensity of each pixel along the scan line must be determined.

In determining the intensity value for each pixel, we must first determine the intensity of each polygonal vertex. This is done by substituting the surface normal with a vertex normal in the illumination model. The vertex is the average of the surface normals for all polygons sharing that vertex as illustrated in Figure 2.4. After the intensity of that vertex is determined, a bilinear interpolation is applied to the intensity at the vertices. [Ref.4:pp. 40,41]

a. Determining the Intensity at a Point

Figure 2.5 demonstrates the interpolation scheme used in Gouraud's technique. As stated previously, first we must determine the intensity at the vertices of each polygon. Once this has been accomplished, the intensity of all other points in the polygon can be determined. In determining the intensity of point **Z,** we must first determine the intensity value of points **J** and **K.** The intensity value for point **J** is determined by taking the intensity values of points **A** and **B** an interpolating the value of **J** with the following calculations. [Ref.4:pp. 44]

$$I_J = \frac{|\overline{JB}|}{|\overline{AB}|} I_A + \frac{|\overline{AJ}|}{|\overline{AB}|} I_B$$

where

- I_A is the intensity at vertex A
- I_B is the intensity at vertex B
- I_J is the intensity at point J

The intensity at K, I_K is computed similarly.

$$I_K = \frac{|\overline{KC}|}{|\overline{AC}|} I_A + \frac{|\overline{AK}|}{|\overline{AC}|} I_C$$

The intensity at Z, I_Z, is then obtained by interpolation.

$$I_Z = \frac{|\overline{ZK}|}{|\overline{JK}|} I_J + \frac{|\overline{JZ}|}{|\overline{JK}|} I_K$$

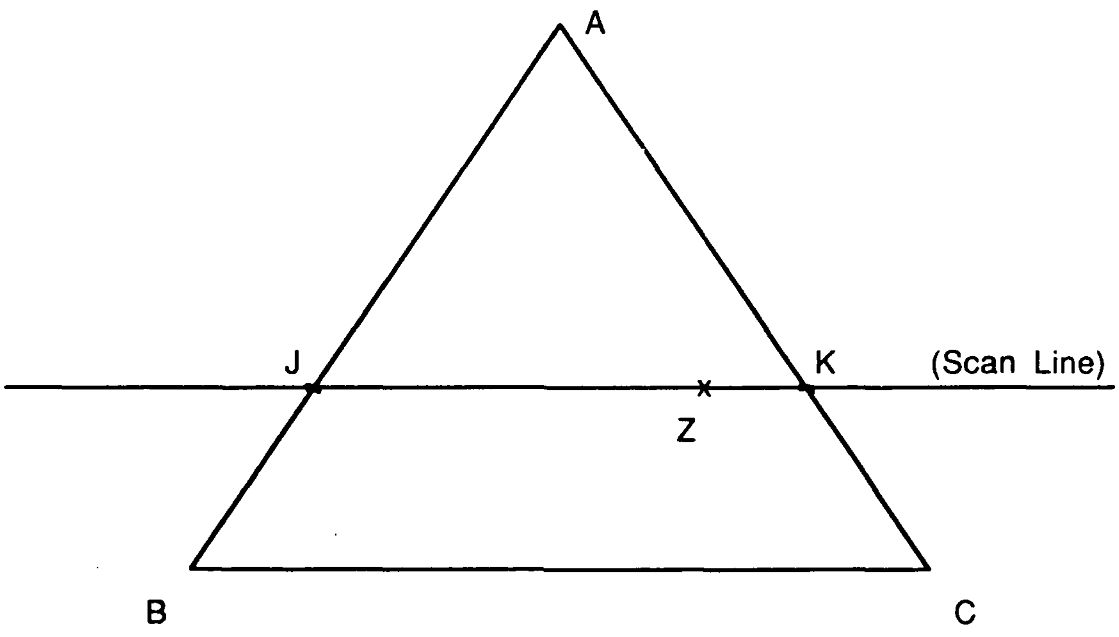


Figure 2.5 Interpolation Scheme

b. Performance

Gouraud shading does an excellent job in removal of intensity discontinuities between adjacent planes of surface representation. It handles the problems experienced in the constant intensity method quite well, however, it does have some problems of its own. One such problem is Mach banding, which is the appearance of light or dark streaks on the surface of an object caused by sharp edges between polygons. This streaking occurs because Gouraud's technique handles discontinuity of intensity across the boundaries of adjacent polygons but does not address the continuity of change in intensity over the surface. Another problem that exists in Gouraud's technique is what one might describe as a case of Gouraud shading working too well. In this we are referring to the situation where there are two or more adjacent polygons that may not be on the same plane but have vertex normals that are the same, causing the surface to appear flat [Ref.4:pp. 46]. This problem can be corrected by manually selecting the vertex normals such that when vertex A is being used with polygon 1, the normal is different than when vertex A is being used with polygon 2. Although there are a few problems with Gouraud Shading, it is a quantum leap over constant intensity.

F. IMPLEMENTATION

While the lighting system on the IRIS-4D/70G provides a more realistic image, it is not without cost. In certain case, a scene's update rate, that is not using the lighting system, has been reduced by as much as one half when it uses the lighting system. With the implementation of any system or algorithm there exists a very high probability of encountering some problems, and the implementation of the IRIS-4D/70G's lighting system is no exception.

1. Dead Spot

At certain viewing positions under certain conditions, a polygon can appear to enter what we call a dead spot. For the purpose of this study, a dead spot is defined as a position where a polygon should be reflecting light but is not, giving the appearance of an object that has no light source. This phenomenon appears only when the following conditions are met: (1) a constant intensity shade model is used, (2) the light vector is perpendicular to the polygon's surface, and (3) the view point is perpendicular to the polygon's surface. This problem can be avoided by ensuring that the three conditions required are not achieved.

2. <u>Color</u>

In determining various colors other than red, green, blue, black, and white, while utilizing the lighting system on the IRIS-4D/70G, a great deal of trial an error is used. When trying to derive a color using the RGB color scheme, without using the light system, one simply mixes a certain amount of red, green, and blue together. This is easily done with the aid of a programming tool called COLORS[1]. This program allows the user to mix various amounts of red, green, and blue together until the desired color is achieved. These colors range in value from 0 to 255. It is well known that black is absence of color or in the case of the programming aid the value zero for all three colors, while white is the presence of all colors with the value 255 for all three colors. Various shades of grey can be represented by having the red, green, and blue values all equal the same, i.e., 155 for red, 155 for blue, and 155 for green. With this in mind, a new programming tool was created that takes the lighting and shading capabilities of the IRIS-4D/70G into consideration.

The lighting system of the IRIS-4D/70G has a number of variables that are grouped together into the following three categories: Material Property, Light Property, and Light Model. When attempting to define a specific color, it is the material property we are most interested in. The material property consists of all the properties used to define the surface characteristics of a material.

While using the COLORS program to determine the color of a surface, there are only three variables to be concerned with (ie. red, green, and blue values). As can be

[1]This program was created by Jonathan Bowen of SGI.

seen in Figure 2.6, when determining a specific color while using the lighting system there are 13 variables that can directly affect how the color of an object is viewed. Unlike the variables in the COLORS program, the variables in the lighting program do not provide the colors as one may think. In trying to determine simple colors, a great deal of trial and error is used. A solution to this problem is to create a library or file of the most commonly used surface materials such as copper, gold, etc., for future use. This still does not prevent a long process of defining the surface materials initially.

3. Viewing the Backside of an Object

While at sea as part of a Battle Group, most vessels are often flanked on one or both sides by other vessels. A captain of a vessel may look to his right and find a vessel along with the sun in his field of view. When the situation of an object being positioned between the viewer and the light source occurs on the IRIS-4D/70G, the object appears to be in the form of a silhouette. This occurs because the normal vectors of the side closest to the viewer are pointed away from the light source. If the object is located relatively far away from the viewer, then this appearance is acceptable. If the object is located relatively close to the viewer then this condition presents an inaccurate representation of the object.

This problem can be solved by determining a color that is very close to the color of the object but with less intensity. Once this color has been determined, it will be assigned as the object's emission color. If the emission color is the same color as the reflecting color but with less intensity, it will only be seen when the intensity of the object's reflected light decreases to a level that is less than or equal to the intensity level of the emitted color.

III. HIDDEN SURFACE TECHNIQUES

Hidden surface techniques are algorithms that attempt to determine the surfaces and edges that are visible and invisible to a viewer at a specific point. The removal of the surfaces and edges that should not be seen by the viewer is one of the most difficult problems in the field computer graphics. There is no one best solution to this problem. There exist a tremendous number of hidden surface techniques available today [Ref.5:pp. 189]. In this study, some of these techniques are studied. Given below is a brief discussion of these techniques, along with some of the advantages and disadvantages as they relate to real-time images.

A. Z-BUFFERING

Z-Buffering, also known as depth-buffering, is one of the simplest hidden surface removal algorithms. This algorithm determines the visibility of a scene one pixel at a time. It only draws the pixel with the smallest z value as illustrated in Figure 3.1. This value is determined from a pixel by pixel comparison of the entire scene.

1. Performance

The z-buffer technique, from a user's point of view, is one of the easiest hidden surface techniques to implement. One of the greatest features about the z-buffer technique is, the user does not have to be careful of such things as drawing order and what order should the vertices be placed in. All of this tedious work is done by special hardware in the IRIS-4D/70G. Although the z-buffering technique is very easy to implement, it has seldom been considered in the creation of large animated scenes. This is due solely to the lack of available workstations that could provide an acceptable z-buffered polygon fill rate that could achieve real-time performance. As stated previously,

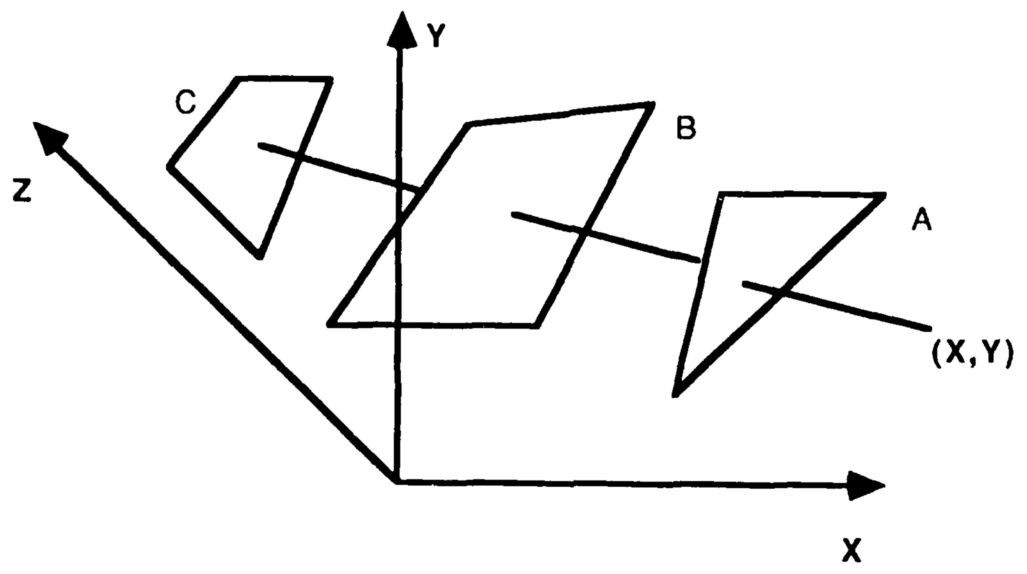


Figure 3.1 Z-Buffering

this technique must do a pixel by pixel comparison of the entire scene before it can be displayed and this is a time consuming process. With the arrival of the IRIS-4D/70G, with an advertised z-buffered polygon fill rate of 5,500 polygons per second, [Ref.2] this technique is greatly enhanced. Although, the ability of the z-buffer technique to achieve real-time performance is greatly enhanced, it still lacks the ability to handle very large scenes in real-time.

B. BACKFACE POLYGON REMOVAL

Another common method of hidden surface removal is the backface removal algorithm. This algorithm determines the backface of the surface by determining the rotation direction or drawing order that the vertices of a polygon are drawn in. If the polygon's vertices are drawn in a counter clockwise rotation, then the polygon is drawn. If the polygon's vertices are drawn in a clockwise rotation, then the polygon is not drawn as illustrated in Figure 3.2.

1. Performance

On simple images, this algorithm works well and is relatively easy to implement. However, when the scene's complexity increases, such as the case in the design of our ship model, three problems that are barely noticeable in simple images such as Figure 3.2, become quite noticeable. One such problem that exists is, the user must keep track of the order in which the vertices are drawn in a polygon to ensure that the side visible to the viewer is the side that is actually desired. During the design of our ship model, this problem was considered to be a trivial one that would have very little effect on the construction of our model. Prior to constructing the model a side view and a top view were drawn out so a better view of the model would be available. There were no problems in constructing the side that had been drawn out, however a problem did exist when constructing the side that could not be viewed. A number of polygons were drawn in the wrong rotation order causing certain polygons to be visible when they should have been invisible and vice versa. This problem is easily solved by locating the incorrect polygon and changing the drawing order of the vertices but this task becomes tedious and time consuming. Another problem experienced was the appearance of gaps between polygons that were adjacent and shared adjoining vertices. This problem occurs when adjacent polygons with abrupt changes are viewed from a certain position. This position is normally just after the adjacent polygon comes into view. To solve this problem, the

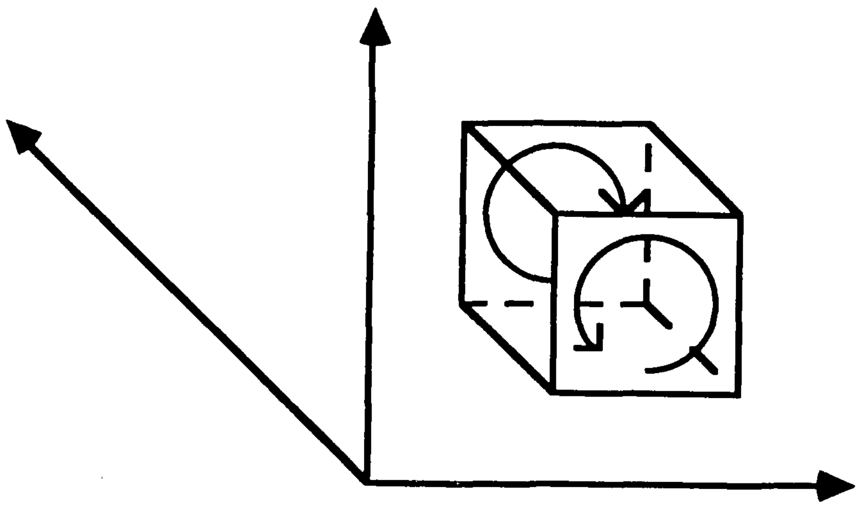


Figure 3.2 Backface Polygon Removal

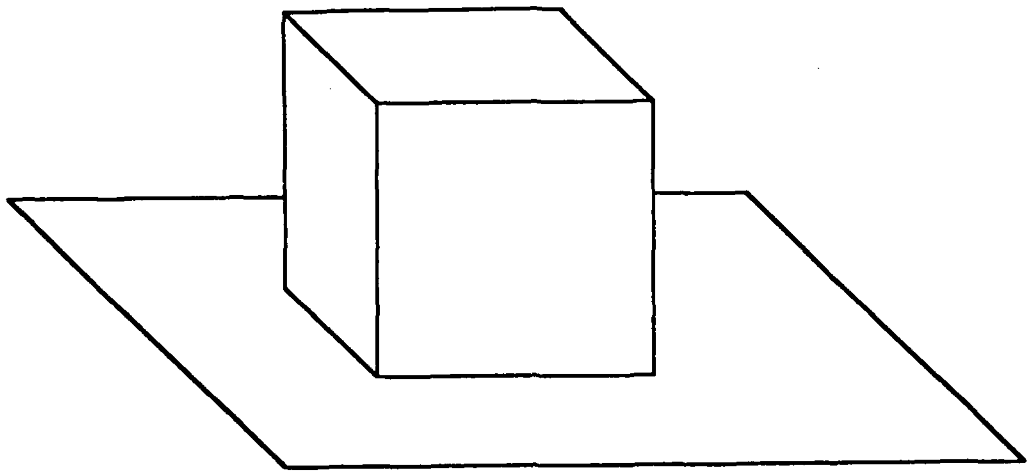
adjacent polygons can be design to overlap each other slightly. Although this procedure solves the problem, it is not used because of possible side effects in computing vertex normals, which will be discuss later in this study. Lastly, when creating certain images, backface removal must be used in conjunction with other hidden surface removal technique, such as painter's algorithm, to accurately depict the scene. Continuous attention to the drawing order of the polygons is a must when using this technique.

Figure 3.3a illustrates how a box on top of a flat surface should appear. In this illustration, the flat surface is drawn first followed by the box. Figure 3.3b shows what happens when the drawing order in the scene is not correct. In this illustration, the box is drawn first, followed by the flat surface.

When used appropriately, in simple scenes or in conjunction with other hidden surface removal techniques, backface removal is a very powerful hidden surface removal technique. It is computationally efficient allowing for real-time animation, however, because it can not accurately represent complex scenes alone, this computation efficiency may decay in certain application areas. Because of the numerous problems encountered with the backface removal technique, it was abandoned in favor of the z-buffer technique.

C. SUMMARY

There exist several other hidden surface removal techniques and trying to determine which one is the best one overall is a difficult if not an impossible task. The effectiveness of a hidden surface removal method depends on the characteristics of a particular application. If the surfaces in a scene are spread out in the z direction so that there is very little overlapping in depth, a depth-sorting method may be best. For scenes with surfaces fairly well separated horizontally, a scan-line method may be best [Ref.3:pp. 272]. Therefore, the performance of the hidden surface removal method is dependent on the application in which it is to be used.


3a. Correct drawing order

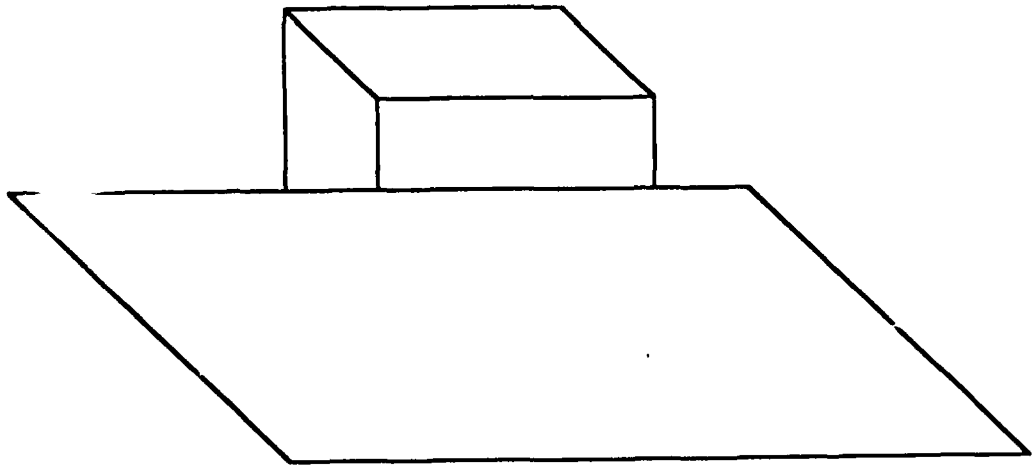

3b. Incorrect drawing order

Figure 3.3 Polygon Drawing Order

IV. A THREE-DIMENSIONAL VISUAL DISPLAY

It is believed that the addition of three-dimensional computer graphics to the Command and Control Workstation of the Future (CCWF) will be a tremendous asset to today's commander. Today's commander, unlike his predecessors of World War II, does not enjoy the luxury of having ample time to interpret the data before him and make his decisions. This is due mainly to jet propelled aircraft and cruise missiles that can effectively engage a vessel or command post from long ranges in a matter of seconds. In an environment such as this, a second can mean the difference between survival or destruction. A three-dimensional real-time animated display will allow the commander to be able to interpret thousands of bytes of information very rapidly, allowing him to make more accurate and rapid decisions.

A. PROBLEMS

Although a three-dimensional real-time display is very easy to interpret, it is very difficult to construct. One of the major problems in building our three-dimensional display is providing a unified data format for various components of the three-dimensional display, i.e., the object data, the material data, and the lighting data.

B. OBJECT DATA FORMAT

The format chosen[2] for representation of objects allows for a minimum number of lines to be used to store the object, while allowing for relatively easy direct editing. All types of polygons are supported, although actual rendering of concave polygons depends

[2] An example of this object data format is contained in Appendix C.

on hardware support. We review why each piece of data is required and how it is obtained.

1. <u>Material</u>

Material is the type of material the polygon is made of. The actual material name follows the command Material and acts as an index to a material property library. The material name is not case sensitive. This name is matched with the name of a material in the material property library. Once a match has occurred, the matching material is used to define all polygons that follow it until a new material has been defined. If there is no match, then an error message will occur stating the problem and execution will terminate. The materials in our material property library are defined by specifying the coefficient values of the emitted light, ambient light, diffuse light, and specular light. The emitted and ambient light coefficients model the intensity of the emitted and ambient light. The specular and diffuse light coefficients model the percentage of incident light reflected specularly and diffusely. Each coefficient has a red, green, and blue component that varies from 0.0 to 1.0 in value. A material scattering exponent is also specified to determine how shiny the surface is.

2. <u>Snorm</u>

Snorm is used to determine when a unit surface normal is used. This unit surface normal is utilized in shading models such as constant intensity. It consists of x, y, and z coefficients for the vector. The unit surface normals are provided in the object data file.

3. <u>Vnorm</u>

Vnorm is used to determine when a unit vertex normal is used. The unit vertex normal is utilized in the Gouraud shading model. The vertex normal, which is an x, y, and z coefficient for each vertex vector, is provided in the object data file.

4. Polygon Vertices

A graphics object is a collection of polygons. Each polygon is defined by three or more vertices. Each vertex is defined by an xyz coordinate. The polygon vertices of the object are provided in the object data file.

C. DRAWING THE MODEL

In verifying our file format, we constructed a ship model. This model is designed like a subroutine and can be called as one. The model was drawn using the z-buffering technique. Each polygon is drawn by first defining the color of the material it is to be made of, then by defining its vertices, then by defining its normal(s), and then drawing the polygon using graphics calls to polygon drawing functions.

1. Scale

In creating the visual display, several factors had to be considered. One being the establishment of some sort of standardization in terms of creating various ship models. In order for an aircraft carrier and a destroyer to be depicted realistically when they are positioned side by side, they must be drawn using the same scale. It was decided that the models would be created in meters. The model is created in meters because although some publications will provide the measurements in both the English and metric system, most publications will only produce the measurements in meters. This is especially true of foreign vessels.

2. Amount of Detail

The amount of detail displayed in an image is a direct result of the number of polygons used to create that image. Prior to creating the ship model, a study was conducted on the IRIS-4D/70G to determine the maximum number of polygons our model could be constructed of. One of the main considerations in this study is to be able to draw the scene while maintaining approximately 5-6 frames per second. For the purpose of this study only, a scene is defined as a view from the bridge of a ship where a maximum of four ships would be in view along with the ocean. The number four is chosen because it is believed that from any one view point only a maximum of four ships would be seen in the same general area for various reasons. The results of the study indicated that an image of approximately 1000 polygons using the z-buffering hidden surface removal technique and a lighting model, has a frame update rate of approximately four frames per second. Although this does not reach our intended goal of five to six frames per second, it is acceptable for our application. When using approximately 1000 polygons to construct a scene, the ship models can be constructed with approximately 250 polygons each, providing a great degree of detail. The water doesn't cost anything since it can be created of one polygon.

3. Normals

One of the most important features when using a lighting model is the determination of the normal. If the normals are determined incorrectly, the image will be inaccurately depicted. There are several problems that can contributed to incorrect normals. One such problem being, if the normal vector is determined incorrectly, such that the vector is pointing away from the light as illustrated in Figure 4.1, then that polygon will appear either black or only the emitted light from that polygon will be visible.

D. COMPUTING THE NORMALS

Determining the normals of a surface area is a tedious and difficult task. The need for a tool to assist in determining these normals is apparent. ADDNORM is a program that we developed to compute the normals of various polygons. This program is written in the C programming language. ADDNORM uses as input, a file containing the

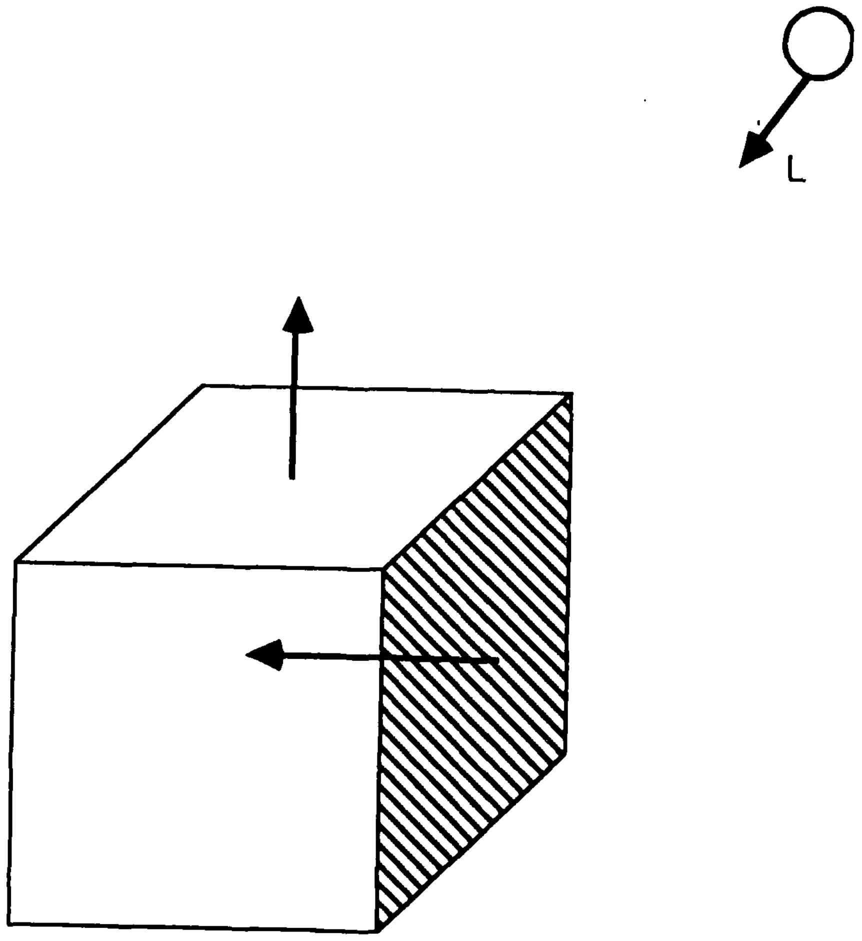


Figure 4.1 Incorrect Normal

polygon's vertices and other pertinent information required to compute the normals. We review why each piece of data is required for the computations and how it is obtained.

1. Inside-Pt

The inside point of a graphics object is a point in the interior of a graphics object that is used to determine the correct orientation of the normal, i.e., perpendicular to the surface and pointing away from the interior point. Orientation of a surface, the outward facing surface of the polygon, is determined by this normal vector. Some objects, such as our ship model, are made of several smaller objects or subobjects. In this case a different inside point is required for each subobject. The inside point is an xyz coordinate and is used as input for our ADDNORM program.

2. Area

The command Area is used to separate different surface areas of a graphics object. For the purpose of this study, the area of a graphics object is a surface of an object that is created of as few as one polygon or as many polygons as the system will allow. Vertex normals are computed by averaging the unit surface normals of its surrounding polygons. The command Area is used to define the surrounding polygons and is discussed in greater detail later in this chapter.

3. Snorm and Vnorm

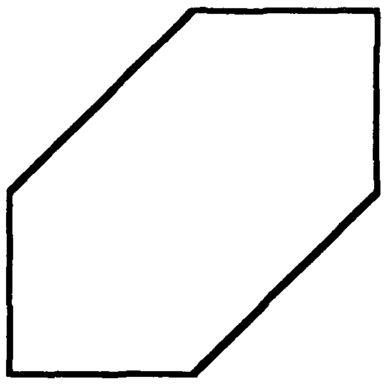
The Snorm and Vnorm commands provide the user with the means of selectively choosing which polygons use unit surface normals and which polygons use unit vertex normals. Snorm is used to determine when a unit surface normal is to be computed. Vnorm is used to determine when a unit vertex normal is to be computed.

4. Polygon Vertices

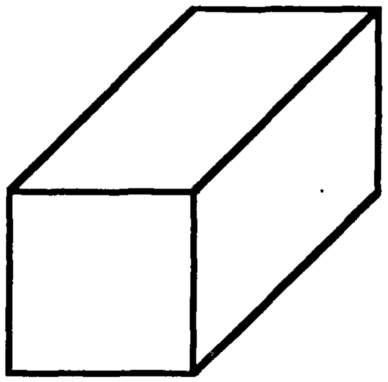
The polygon vertices, which are defined by an xyz coordinate, are directly used in the computation of the normals.

5. Computing Normals using ADDNORM

When computing the normals, ADDNORM takes several conditions into account. One of the conditions it takes into account is what kind of normal is to be computed, unit surface normal or vertex normal. This is determined by the Snorm and Vnorm commands. Computation of vertex normals presents a problem. The vertex normal can be an average of the unit surface normals of the surrounding polygons or it can be equal to the unit surface normal of the polygon itself. A problem encountered



Averaging Surface Normals

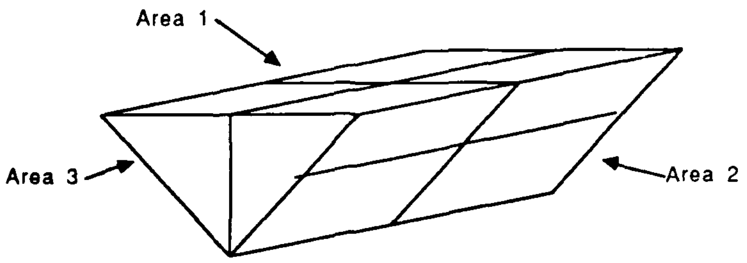


Vertex Normals equal to Surface Normal

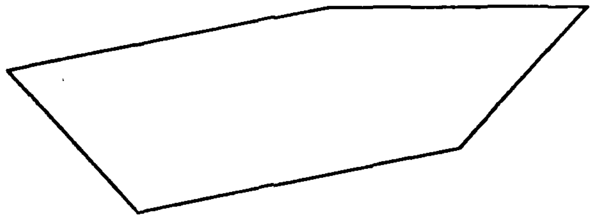
Figure 4.2 Methods of Computing Vertex Normals

when computing vertex normals is the representation of sharp edges on an object. This can best be described by using a cube. When a cube is rendered, if the vertex normals are derived by averaging the surrounding unit surface normals then the edges will not be visible, as illustrated in Figure 4.2. If the vertex normals are derived by allowing the vertex normal to be equal to the unit surface normal, then a correct representation of the cube is achieved. Although this procedure works ideally for an object as simple as a cube, in the case of more complex objects, modifications to this technique are required.

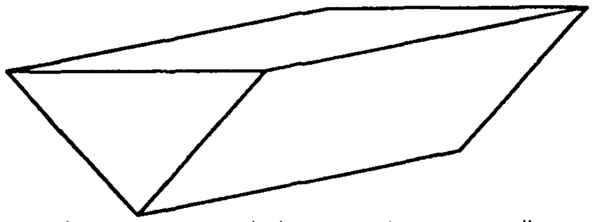
A technique that is commonly used in drawing more complex objects is the creating of surfaces of an object by using several polygons to represent that surface. In the design of our ship, the right side of the hull is created of several polygons, the deck is also created of several polygons. These two surfaces adjoin alone a common edge. Although we want the surface of the right side of the hull to appear as one continuous smooth surface, we also want the adjoining edge between the deck and the right side to be distinguishable. This is achieved by the command "AREA". This command marks the beginning of an area to be used for computing the vertex normals. The polygon in which the vertex normals are computed for, is a member of a set of polygons defined by the command Area and only those polygons within that set are used as surrounding polygons. The set can be as small as one polygon, in which case the vertex normal is equal to the unit surface normal, or it can be as large as the particular hardware support will allow. The polygons used to define a specific area are those that follow the command AREA. These polygons are used until either another area is defined by the appearance of the command AREA again or the end-of-file command is reached. By dividing an object into various surface areas, as illustrated in Figure 4.3a, we can achieve the desired results as illustrated in Figure 4.3c. Once the normals have been computed, ADDNORM stores them along with the polygon vertices and all other necessary



4.3a Creating surfaces using several polygons



4.3b Computing vertex normals by averaging all surrounding polygons



4.3c Computing vertex normals by averaging surrounding normals within their own area.

Figure 4.3 Computing Vertex Normals using ADDNORM

```
#include "gl.h"
buildship()

{
/* HULL */

/* STARBOARD SIDE */

/* ABOVE THE WATERLINE */

/* AFT SECTION */

/* This polygon is drawn using vertex normals */

xyznormal(-0.9941,0.0426,0.0994);
pmv(-70.00,3.00,7.00);
xyznormal(-0.9983,0.0000,0.0587);
pdr(-68.00,0.00,4.00);
xyznormal(-0.9942,0.0000,0.1077);
pdr(-60.00,0.00,6.50);
xyznormal(-0.9889,0.0494,0.1401);
pdr(-60.00,3.00,8.50);
pclos();

/* This polygon is drawn using surface normals */
/* AFT OF HULL */
pmv( -70.00,   3.00,  -7.00);
pdr( -68.00,   0.00,  -4.00);
pdr( -68.00,   0.00,   4.00);
pdr( -70.00,   3.00,   7.00);
xyznormal(-0.8321,-0.5547,0.0000);
pclos();
}
```

Figure 4.4 Output from OBJMAKER

information into a file for further use. With the proper input, ADDNORM's output is in the format of our object data file.

E. OBJMAKER

The OBJMAKER is a program that is written in the C programming language that takes as input a file that has both polygon vertices and normals in it. This program uses the object data file as input. The program takes the various polygon vertices and their normals and provides the appropriate drawing commands for them. Figure 4.4 is an example of the output generated by the program OBJMAKER.

Both program ADDNORM and program OBJMAKER are designed with one primary function in mind. That function is to aid the programmer in the creation of graphics objects. ADDNORM and OBJMAKER are both executable files that require only a small amount of time to perform their tasks.

V. CONCLUSION AND RECOMMENDATIONS

Determining the optimum trade-offs required to achieve a realistic looking image while maintaining real-time or near real-time performance is a difficult task. This is so because there are so many variables that effect the outcome. Variables such as the lighting model, hidden surface techniques, and amount of detail to be displayed.

In determining the amount of detail our scene displays, we chose to build ships that provided more detail. By making this decision, we chose realism over real-time in this case. The ship model selected is one of complex structure, an Aegis class destroyer. It was selected because of its complex structure. The design of this ship model is an attempt to represent a worse case in detail design. To provide the CCWF with a detailed display while maintaining a more real-time performance, we must be very selective with our model types. Our ship model is created of approximately 250 polygons. Of this 250 polygons, approximately one third was used to draw the hull and the rest was used to draw the super structure. An aircraft carrier because of its simple super structure can be designed with almost one half the number of polygons used to draw our model. A tanker or cargo ship can be drawn with a relatively fewer number of polygons also. Although it would be nice for the CCWF to have a graphics library that consists of a large number of ships, this is not necessary. A graphics library consisting of three different ships, a large military combatant (aircraft carrier), a regular combatant (frigate), and a merchant ship (super tanker), could be used to depict just about any surface scenario at sea. A scene using these three models could be used to accurately display a scene and maintain better real-time performance than a model as complex as ours.

A. LIMITATIONS

The program ADDNORM provides an easy method for computing vertex normals, however, it has some drawbacks. Although it handles the computation of vertex normals well, there may exist situations that this tool is not designed to cover. Another drawback is, ADDNORM requires the user to pay close attention to the grouping of the polygons, i.e., a polygon of a particular area must be grouped together for the correct vertex normal to be computed. This may be in direct conflict with the grouping order required when using other hidden surface techniques such as backface removal.

B. FUTURE RESEARCH

An area of further research in this study that is greatly needed is the development of a graphics material library for the lighting system on the IRIS-4D/70G. In creating this library, the development of a technique to aid in determining a specific material type will be very valuable. At the present time, determining a specific material type is strictly by trial an error and is very time consuming.

Another possible research area is to determine better techniques for creating complex models such as through the use of digitized models. Maybe this work could allow for a ship model to be converted by a digitized camera into a three-dimensional image that could be used in real-time.

APPENDIX A - PROGRAM ADDNORM

```
#include "stdio.h"
#include "math.h"

#define STRINGSIZE 81
#define VERTEX     0
#define SURFNORM   1
#define X          0
#define Y          1
#define Z          2
#define MAX_AREAS 200
#define MAX_POLY  500
#define MAX_COORD  15
#define XYZ        3

int alphabet(),digit();

main()
{
  char s[81],garbage[81];
  int i, j;

  char c;
  long num_of_coord;
  float xcoord,ycoord,zcoord;

  float xyz[1000][3];
  float vertnormal[15][3];
  float surfnormal[3];

  float coord_surfnorm[MAX_POLY][MAX_COORD][2][XYZ];

  float xinside, yinside, zinside;

  long area_number;              /* Counts the number of different */
                         /* surface areas on an object    */

  long num_of_poly;              /* Counts the number of polygons on */
                         /* a specific surface area.      */

  long poly_count;             /* Count the total number of polygons */
                        /* in an object.                */

  long max_poly_count[200];       /* Holds the value for the number of */
                         /* polygons on each surface area. */

  int area_marker[MAX_AREAS];      /* An array that marks the beginning */
                        /* of each new area.             */
```

```
FILE *input_file, *output_file, *fopen();

input_file = fopen("shipcoord","r");

if (input_file == NULL)
{
 fprintf(stderr,"0annot open this file!!!0);
 exit(1);
}
else
{
 area_number = -1;
 poly_count = 0;
 num_of_poly = 0;
 while (((c = getc(input_file)) != EOF) && ((c != 'H') && (c != 'h')))

 /* Continue until an EOF is reached or a HALT command is reached */

 {
  if((c == 'A')||(c == 'a'))
  /* Check for AREA command */
  {
   if (area_number >= 0)
   {
    /* Store the number of polygons  that the last area is made of */
    max_poly_count[area_number] = num_of_poly;

    /* Marks the beginning of a new area */
    area_marker[area_number + 1] = poly_count;

   }
   else if (area_number == -1)
    area_marker[0] = 0;         /* Initilize the first area */

   area_number = area_number + 1;
   num_of_poly = 0;              /* Reset the polygon counter to zero *;
  }
  else if (c == '#')
  {
   fgets(garbage,STRINGSIZE,input_file);
  }
  else if ((c == 'I')||(c == 'i')) /* Check to see if this is the */
                                  /* inside point or reference pt */
                                  /*used by the compute normal routines*/
  {
   fgets(garbage,STRINGSIZE,input_file);
   fscanf(input_file,"%f %f %f",&xinside,&yinside,&zinside);
  }
  else if (digit(c))
  {
     ungetc(c,input_file); /* push character back in front of*/
```

```
                        /* pointer so it can be read as a */
                        /* decimal instead of a character */

        fscanf(input_file,"%d",&num_of_coord);

        fgets(garbage,STRINGSIZE,input_file);
        for (i = 0; i < num_of_coord ; i = i+1)

        {
         fscanf(input_file,"%f %f %f",&xyz[i][0],&xyz[i][1],&xyz[i][2]);
        } /* end for stmt */

        computesurfnormal(num_of_coord,xyz,xinside, yinside, zinside,
                    surfnormal);

        if (area_number < 0)
          area_number = 0;

        for (i = 0; i < num_of_coord ; i = i+1)
        {
         coord_surfnorm[poly_count][i][0][0] = xyz[i][X];
         coord_surfnorm[poly_count][i][0][1] = xyz[i][Y];
         coord_surfnorm[poly_count][i][0][2] = xyz[i][Z];

         coord_surfnorm[poly_count][i][1][0] =
                  surfnormal[X];
         coord_surfnorm[poly_count][i][1][1] =
                  surfnormal[Y];
         coord_surfnorm[poly_count][i][1][2] =
                  surfnormal[Z];

        } /* end for stmt */

        num_of_poly = num_of_poly + 1; /* counts the number or polygons */
                          /* in a particular area */

        poly_count = poly_count + 1;  /* counts the total number of poly*/
                          /* in the object. */

      } /* end else if stmt */

     } /* end while stmt */

    if (c == EOF)
     max_poly_count[area_number] = poly_count;

   } /* end else stmt */

   fclose(input_file);

   input_file = fopen("shipcoord","r");
```

```
area_number = -1;

if (input_file == NULL)
{
 fprintf(stderr,"0annot open this file!!!0);
 exit(1);
}
else
{
 output_file = fopen("objdata","w");

 while (((c = getc(input_file)) != EOF) && ((c != 'H') && (c != 'h')))
 {

  if (c == '#') /* This particular character will allow for the */
             /* entire line to be copied into the file without */
             /* any alterations. */

  {
   ungetc(c,input_file);
   fgets(s,STRINGSIZE,input_file); /* read the entire line */
   fputs(s,output_file);          /* copy the entire line */
  }

  else if ((c == 'I')||(c == 'i')) /* Check to see if this is the */
                          /* inside point or reference pt */
                          /* used by the compute normal routines*/

  {
   fgets(garbage,STRINGSIZE,input_file);
   fscanf(input_file,"%f %f %f",&xinside,&yinside,&zinside);
  }

  else if ((c == 'A')||(c == 'a'))

  {
   area_number = area_number +1;
  }

  else if (alphabet(c))

  {
   if ((c == 'V')||(c == 'v')) /* check to see if vertex normals */
                        /* are being used.*/
   {
    /* collect any unread characters on the line until an end*/
    /* of line marker is incountered */

    ungetc(c,input_file);
    fgets(s,STRINGSIZE,input_file);
    fputs(s,output_file);
   c = getc(input_file); /* read first character in line */

   if (digit(c)) /*check to see if character is a number */
   {
    ungetc(c,input_file); /* push character back in front of*/
                     /* so it can be read as a decimal */
                     /* instead of a character */
    fscanf(input_file,"%d",&num_of_coord);
    fprintf(output_file,"%d0,num_of_coord);

    fgets(garbage,STRINGSIZE,input_file);

    for (i = 0; i < num_of_coord ; i = i+1)
    /* read the vertices and their normals and write them*/
    /* in the selected file */
     {
      fscanf(input_file,"%f %f %f",&xyz[i][0],&xyz[i][1],
          &xyz[i][2]);
    } /* end for stmt */
    if (area_number < 0)
      area_number = 0;

    computevertnormal(num_of_coord, max_poly_count[area_number],
               xyz, coord_surfnorm, area_marker[area_number],
               vertnormal,xinside,yinside,zinside);

    for (i = 0; i < num_of_coord ; i = i + 1)
     {
      fprintf(output_file,"%7.2f %7.2f %7.2f %8.4f %8.4f %8.4f0,
           xyz[i][0],xyz[i][1],xyz[i][2],vertnormal[i][0],
           vertnormal[i][1],vertnormal[i][2]);
     }

    fprintf(output_file,"0);
   } /* end if stmt */

   else
   {
    fprintf(stderr,"Data FORMATTED incorrectly0);
    exit(1);
   }
  } /* end if stmt */
  else if ((c == 'S')||(c == 's'))
  {

   ungetc(c, input_file);
   fgets(s,STRINGSIZE,input_file);
   fputs(s,output_file);
   c = getc(input_file); /* read first character in line */

   if(digit(c)) /*check to see if character is a number */
```

```
          {
           ungetc(c,input_file); /* push character back in front of*/
                         /* so it can be read as a decimal */
                         /* instead of a character */
           fscanf(input_file,"%d",&num_of_coord);
           fgets(garbage,STRINGSIZE,input_file);
           fprintf(output_file,"%d0,num_of_coord);

           for (i = 0; i < num_of_coord ; i = i+1)
           /* read the vertices and  write them*/
           /* in the selected file adding the appropiate normals*/
           /* to them */
           {
            fscanf(input_file,"%f %f %f",&xyz[i][0],&xyz[i][1],&xyz[i][2]);
            fgets(garbage,STRINGSIZE,input_file);
           } /* end for stmt */
           computesurfnormal(num_of_coord,xyz,xinside, yinside, zinside,
                        surfnormal);

           for (i = 0; i < num_of_coord ; i = i + 1)
           {
            fprintf(output_file, "%7.2f %7.2f %7.2f0, xyz[i][0], xyz[i][1],
                  xyz[i][2]);
           }

           fprintf(output_file,"%8.4f %8.4f %8.4f0, surfnormal[0],
                 surfnormal[1], surfnormal[2]);


          } /* end if stmt */

          else
          {
           fprintf(stderr,"Data FORMATTED incorrectly");
           exit(1);
          }
        } /* end else if stmt */
       } /* end else if alphabet stmt */
       else if (c == ' ')
       {

       }
     } /* end WHILE stmt */
    } /* end else stmt */
  } /* end main */

  int alphabet(x)
  char x;
  {
   if ((x >= 'A' && x <= 'Z') || (x >= 'a' && x <= 'z'))
    return(1);
  else
    return(0);
}

int digit(x)
char x;
{
 if ((x=='0')||(x=='1')||(x=='2')||(x=='3')||(x=='4')||(x=='5')
     ||(x=='6')||(x=='7')||(x=='8')||(x=='9'))
   return(4);
 else
   return(0);
}




/* Author: Professor M. J. Zyda */
/* Module: Computesurfnormal */


/* this function computes the normal given a center
   point...
*/

#include "math.h"
#include "stdio.h"

computesurfnormal(ncoords,xyz,xinside,yinside,zinside,normal)

long ncoords;  /* numbe rof coords in the polygon */

float xyz[][3];

float xinside, yinside, zinside;

float normal[3];  /* returned normal */

{

  long i,j;  /* loop temps */

  float a[3], b[3]; /* vector hold locations for the vectors that run
                    from coordinate 1 to points 0 and 2 of the
                    polygon */

  float xn[3], xmn[3];  /* points on line containing normal that are
                        on opposite sides of the plane containing
                        the polygon.
                      */
```

```
float distton;       /* distance to point n from pt inside. */

float disttomn;      /* distance to point -n from pt inside. */

float normalmag;  /* magnitude of the normal */


/* compute vector a. It runs from coordinate 0 to coordinate 1 */
for(j=0; j < 3; j=j+1)
{
  a[j] = xyz[0][j] - xyz[1][j];
}

/* compute vector b. It runs from coordinate 2 to coordinate 1 */
for(j=0; j <3; j=j+1)
{
  b[j] = xyz[2][j] - xyz[1][j];
}

/* compute a x b to get the normal vector */
normal[0] = a[1]*b[2] - a[2]*b[1];
normal[1] = a[2]*b[0] - a[0]*b[2];
normal[2] = a[0]*b[1] - a[1]*b[0];

/* divide out the normal by its magnitude to make it a unit */
normalmag = sqrt(normal[0]*normal[0] + normal[1]*normal[1] +
            normal[2]*normal[2]);

if(normalmag > 0.0)
{
  normal[0] = normal[0]/normalmag;
  normal[1] = normal[1]/normalmag;
  normal[2] = normal[2]/normalmag;
}
else
{
  /* leave the normal vector alone...
  */
}

/* compute point n, offset pt from coord 1 in direction of normal */
for(j=0; j < 3; j=j+1)
{
  xn[j] = xyz[1][j] + normal[j];
}

/* compute point -n, offset pt from coord 1 in opposite direction
  from normal.
*/
for(j=0; j < 3; j=j+1)
{
      xmn[j] = xyz[1][j] - normal[j];
    }

    /* compute the distance the inside pt is from point n */
    distton = sqrt((xn[0] - xinside) * (xn[0] - xinside) +
                   (xn[1] - yinside) * (xn[1] - yinside) +
                   (xn[2] - zinside) * (xn[2] - zinside));

    /* compute the distance the inside pt is from point -n */
    disttomn = sqrt((xmn[0] - xinside) * (xmn[0] - xinside) +
                   (xmn[1] - yinside) * (xmn[1] - yinside) +
                   (xmn[2] - zinside) * (xmn[2] - zinside));

    /* if the dist(n) < dist(-n), then n points back towards the
       inside point and is on the same side of the plane as inside.
       a x b is then clockwise.
    */
    if(distton < disttomn)
    {
      /* clockwise must negate the normal */
      normal[0] = -normal[0];
      normal[1] = -normal[1];
      normal[2] = -normal[2];
    }
    else
    {
      /* counterclockwise normal ready to go */
    }

  }




/* this function computes the vertex normals when given a center
   point...
*/

#include "math.h"
#include "stdio.h"

computevertnormal(ncoords,num_of_poly,xyz,coord_surfnorm,area_marker,normal,
            xinside,yinside,zinside)

long ncoords;  /* number of coords in the polygon */

long num_of_poly;

float xyz[][3];
float coord_surfnorm[][15][2][3];

int area_marker ;

float normal[][3];  /* returned normal */

float xinside,yinside,zinside;

{

  long i,j,k;  /* loop temps */

  int num_adj_pts = 0;

  float xn[3], xmn[3]; /* points on line containing normal that are
                      on opposite sides of the plane containing
                      the polygon.
                    */
  float distton;       /* distance to point n from pt inside. */

  float disttomn;      /* distance to point -n from pt inside. */

  float normalmag;  /* magnitude of the normal */

  for(i=0; i < ncoords; i = i+1)
   {

    for(j=area_marker; j < num_of_poly + area_marker; j = j+1)
     {
      for(k=0; k < ncoords; k = k+1)
       {
        if ((coord_surfnorm[j][k][0][0] == xyz[i][0])
          && (coord_surfnorm[j][k][0][1] == xyz[i][1])
          && (coord_surfnorm[j][k][0][2] == xyz[i][2]))
          {
           num_adj_pts = num_adj_pts + 1;
           if (num_adj_pts = 1)
             {
              normal[i][0] = coord_surfnorm[j][k][1][0];
              normal[i][1] = coord_surfnorm[j][k][1][1];
              normal[i][2] = coord_surfnorm[j][k][1][2];
             }
            else
             {
              normal[i][0] = normal[i][0] + coord_surfnorm[j][k][1][0];
              normal[i][1] = normal[i][1] + coord_surfnorm[j][k][1][1];
              normal[i][2] = normal[i][2] + coord_surfnorm[j][k][1][2];
             }
           } /* end if stmt */
       }  /* end for (k) stmt */
     }  /* end for (j) stmt */
   }  /* end for (i) stmt */
for(j=0; j < ncoords; j=j+1)
{
 /* divide out the normal by its magnitude to make it a unit */
 normalmag = sqrt(normal[j][0]*normal[j][0] + normal[j][1]*normal[j][1] +
              normal[j][2]*normal[j][2]);

 if(normalmag > 0.0)
 {
   normal[j][0] = normal[j][0]/normalmag;
   normal[j][1] = normal[j][1]/normalmag;
   normal[j][2] = normal[j][2]/normalmag;
 }
 else
 {
   /* leave the normal vector alone...
   */
 }
}

/* compute point n, offset pt from the coord in direction of normal */
for (i=0; i < ncoords; i=i+1)
{
for(j=0; j < 3; j=j+1)
{
  xn[j] = xyz[i][j] + normal[i][j];
}

/* compute point -n, offset pt from the coord in the opposite direction
   from normal.
*/
for(j=0; j < 3; j=j+1)
{
  xmn[j] = xyz[i][j] - normal[i][j];
}

/* compute the distance the inside pt is from point n */
distton = sqrt((xn[0] - xinside) * (xn[0] - xinside) +
            (xn[1] - yinside) * (xn[1] - yinside) +
            (xn[2] - zinside) * (xn[2] - zinside));

/* compute the distance the inside pt is from point -n */
disttomn = sqrt((xmn[0] - xinside) * (xmn[0] - xinside) +
            (xmn[1] - yinside) * (xmn[1] - yinside) +
            (xmn[2] - zinside) * (xmn[2] - zinside));

/* if the dist(n) < dist(-n), then n points back towards the
   inside point and is on the same side of the plane as inside.
*/
if(distton < disttomn)
{
      /* clockwise must negate the normal */
      normal[i][0] = -normal[i][0];
      normal[i][1] = -normal[i][1];
      normal[i][2] = -normal[i][2];
    }
    else
    {
      /* counterclockwise normal ready to go */
    }
    }

  }
```

APPENDIX B - PROGRAM OBJMAKER

```
#include "stdio.h"

#define STRINGSIZE 81

main()
{
 char s[81],s1[5],garbage[81];
 int i, j;

 char c;
 int num_of_coord;
 float xcoord,ycoord,zcoord;
 float xvert_norm,yvert_norm,zvert_norm;
 float xsurf_norm,ysurf_norm,zsurf_norm;

 int alphabet(),digit(),blanks();

 char blank = ' ';

 FILE *input_file, *output_file, *fopen();

 input_file = fopen("objdata","r");
 output_file = fopen("newburk1.c","w");

 if (input_file == NULL)
 {
  fprintf(stderr,"0annot open file 'SHIPDATA'0);
  exit(1);
 }
 else
 {
  while ((c = getc(input_file)) != EOF)
  {
   if (c == '#') /* This particular character will allow for the */
                 /* entire line to be copied into the file without */
                 /* any alterations. */
   {
    fgets(s,STRINGSIZE,input_file); /* read the entire line */
    fputs(s,output_file);           /* copy the entire line */
   }
   else if (alphabet(c))
   {
    if ((c == 'V')||(c == 'v')) /* check to see if vertex normals */
                                /* are being used.*/
    {
     /* collect any unread characters on the line until an end*/
     /* of line marker is incountered */
```

```
fgets(garbage,STRINGSIZE,input_file);
c = getc(input_file); /* read first character in line */

if (digit(c)) /*check to see if character is a number */
{
 ungetc(c,input_file); /* push character back in front of pointer*/
                /* so it can be read as a decimal */
                /* instead of a character */
 fscanf(input_file,"%d",&num_of_coord);

 fgets(garbage,STRINGSIZE,input_file);
 for (i = 0; i < num_of_coord; i = i+1)
 /* read the vertices and their normals and write them*/
 /* in the selected file adding the appropiate drawing*/
 /* commands with them */
 {
  fscanf(input_file,"%f %f %f %f %f %f",&xcoord,&ycoord,
      &zcoord,&xvert_norm,&yvert_norm,&zvert_norm);
  fgets(garbage,STRINGSIZE,input_file);

  if (i == 0) /*Determine if this is the first point to be drawn*/
          /*If so then the pmv command should be used */
          /*instead of the pdr. */
  {
   fprintf(output_file,"xyznormal(%.4f,%.4f,%.4f);0,
       xvert_norm,yvert_norm,zvert_norm);
   fprintf(output_file,"pmv(%.2f,%.2f,%.2f);0,
       xcoord,ycoord,zcoord);
  }
  else
  {
   fprintf(output_file,"xyznormal(%.4f,%.4f,%.4f);0,
       xvert_norm,yvert_norm,zvert_norm);
   fprintf(output_file,"pdr(%.2f,%.2f,%.2f);0,
       xcoord,ycoord,zcoord);
  }
 } /* end for stmt */
 fprintf(output_file,"pclos();0);
} /* end if stmt */

else
{
 fprintf(stderr,"Data FORMATTED incorrectly");
 exit(1);
}
} /* end if stmt */
else if ((c == 'S')||(c == 's'))
{
 /* collect any unread characters on the line until an end*/
 /* of line marker is incountered */
```

```
    fgets(garbage,STRINGSIZE,input_file);
    c = getc(input_file); /* read first character in line */

    if(digit(c)) /*check to see if character is a number */
    {
     ungetc(c,input_file); /* push character back in front of*/
                    /* so it can be read as a decimal */
                    /* instead of a character */
     fscanf(input_file,"%d",&num_of_coord);
     fgets(garbage,STRINGSIZE,input_file);


     for (i = 0; i < num_of_coord; i = i+1)
     /* read the vertices and  write them*/
     /* in the selected file adding the appropiate drawing*/
     /* commands with them */
      {
       fscanf(input_file,"%f %f %f",&xcoord,&ycoord,&zcoord);

       if (i == 0)  /*Determine if this is the first point to be*/
              /*If so then the pmv command should be used */
              /*instead of the pdr. */
       {
        fprintf(output_file,"pmv(%7.2f,%7.2f,%7.2f);0,
             xcoord,ycoord,zcoord);
       }
       else
       {
        fprintf(output_file,"pdr(%7.2f,%7.2f,%7.2f);0,
             xcoord,ycoord,zcoord);
       }
     } /* end for stmt */
     fscanf(input_file,"%f %f %f",&xsurf_norm,&ysurf_norm,&zsurf_norm);
     fgets(garbage,STRINGSIZE,input_file);
     fprintf(output_file,"xyznormal(%.4f,%.4f,%.4f);0,
          xsurf_norm,ysurf_norm,zsurf_norm);

     fprintf(output_file,"pclos();0);
    } /* end if stmt */

    else
    {
     fprintf(stderr,"Data FORMATTED incorrectly");
     exit(1);
    }
   } /* end else if stmt */
  } /* end else if alphabet stmt */
 } /* end WHILE stmt */
fprintf(output_file,"}0* the following routine calls routine          normal() with 3 args */0);
fprintf(output_file,"xyznormal(x,y,z)0);
fprintf(output_file,"float x,y,z;  /* input normal vector */00);
```

```
    fprintf(output_file,"float tmp[3];  /* array to hold the normal*/0);
    fprintf(output_file,"tmp[0] = x;0mp[1] = y;0mp[2] = z;0);
    fprintf(output_file,"normal(tmp);00);
  } /* end else stmt */
} /* end main */

int alphabet(x)
char x;
{
  if ((x >= 'A' && x <= 'Z') || (x >= 'a' && x <= 'z'))
    return(1);
  else
    return(0);
}

int digit(x)
char x;
{
 if ((x=='0')||(x=='1')||(x=='2')||(x=='3')||(x=='4')||(x=='5')
    ||(x=='6')||(x=='7')||(x=='8')||(x=='9'))
   return(4);
 else
   return(0);
}
```

APPENDIX C - SAMPLE DATA FILE

```
#include "gl.h"
#buildship()
#{
#/* HULL */
#/* STARBOARD SIDE */
#/* ABOVE THE WATERLINE */
#/* AFT SECTION */
Material
haze_grey
vnorm
4
 -70.00   3.00   7.00  -0.8321  -0.5547   0.0000
 -68.00   0.00   4.00  -0.7071  -0.7071   0.0000
 -60.00   0.00   6.50  -0.0371  -0.9278   0.3711
 -60.00   3.00   8.50  -0.1238  -0.5504   0.8256

vnorm
4
 -60.00   3.00   8.50  -0.1238  -0.5504   0.8256
 -60.00   0.00   6.50  -0.0371  -0.9278   0.3711
 -50.00   0.00   8.00   0.0000  -0.9487   0.3162
 -50.00   3.00   9.50  -0.0891  -0.4454   0.8909

vnorm
4
 -50.00   3.00   9.50  -0.0891  -0.4454   0.8909
 -50.00   0.00   8.00   0.0000  -0.9487   0.3162
 -40.00   0.00   9.00  -0.0353  -0.7067   0.7067
 -40.00   3.00  10.00  -0.0891  -0.4454   0.8909

#/* AW MID-SECTION */
vnorm
4
 -40.00   6.00  10.00   0.0000  -0.1644   0.9864
 -40.00   0.00   9.00  -0.0353  -0.7067   0.7067
 -20.00   0.00   9.00   0.0234  -0.6245   0.7807
 -20.00   6.00  10.00   0.0000  -0.1644   0.9864

vnorm
4
 -20.00   6.00  10.00   0.0000  -0.1644   0.9864
 -20.00   0.00   9.00   0.0234  -0.6245   0.7807
   0.00   0.00   9.00  -0.0336  -0.6723   0.7395
```

```
  0.00   6.00  10.00  0.0000 -0.1644  0.9864

vnorm
4
  0.00   6.00  10.00  0.0000 -0.1644  0.9864
  0.00   0.00   9.00 -0.0336 -0.6723  0.7395
 20.00   0.00   9.00  0.0222 -0.6096  0.7924
 20.00   6.00  10.00  0.0000 -0.1644  0.9864

vnorm
4
 20.00   6.00  10.00  0.0000 -0.1644  0.9864
 20.00   0.00   9.00  0.0222 -0.6096  0.7924
 42.00   0.00   9.00  0.0054 -0.6508  0.7592
 42.00   6.00  10.00  0.1935 -0.1613  0.9677

#/* AW FWD SECTION */
vnorm
4
 42.00   6.00  10.00  0.1935 -0.1613  0.9677
 42.00   0.00   9.00  0.0054 -0.6508  0.7592
 52.00   0.00   7.00  0.0300 -0.5143  0.8571
 52.00   6.50   9.00  0.2324 -0.2860  0.9296

vnorm
4
 52.00   6.50   9.00  0.2324 -0.2860  0.9296
 52.00   0.00   7.00  0.0300 -0.5143  0.8571
 62.00   0.00   4.50  0.2137 -0.2999  0.9297
 62.00   7.00   7.00  0.3330 -0.3171  0.8880

vnorm
4
 62.00   7.00   7.00  0.3330 -0.3171  0.8880
 62.00   0.00   4.50  0.2137 -0.2999  0.9297
 66.00   0.00   3.00  0.0000 -0.2425  0.9701
 66.00   7.50   6.00  0.2957 -0.3548  0.8870

vnorm
4
 66.00   7.50   6.00  0.2957 -0.3548  0.8870
 66.00   0.00   3.00  0.0000 -0.2425  0.9701
 75.00   0.00   0.00  0.8944  0.4472  0.0000
 73.00   8.00   4.00  0.3588 -0.5383  0.7626
```

```
vnorm
3
 73.00   8.00   4.00  0.3588  -0.5383   0.7626
 75.00   0.00   0.00  0.8944   0.4472   0.0000
 77.00   3.00   0.00  0.4562  -0.3041  -0.8363

vnorm
3
 73.00   8.00   4.00  0.3588  -0.5383   0.7626
 77.00   3.00   0.00  0.4562  -0.3041  -0.8363
 80.00   7.00   0.00  0.4337  -0.3253  -0.8403

vnorm
3
 73.00   8.00   4.00  0.3588  -0.5383   0.7626
 80.00   7.00   0.00  0.4337  -0.3253  -0.8403
 83.00   9.00   0.00  0.3588  -0.5383  -0.7626

#/* BELOW WATERLINE */
#/* AFT SECTION */
vnorm
4
-68.00   0.00   4.00 -0.7071  -0.7071   0.0000
-67.00  -1.00   3.00  0.0000   1.0000   0.0000
-60.00  -1.00   4.00  0.0000   1.0000   0.0000
-60.00   0.00   6.50 -0.0371  -0.9278   0.3711

vnorm
4
-60.00   0.00   6.50 -0.0371  -0.9278   0.3711
-60.00  -1.00   4.00  0.0000   1.0000   0.0000
-50.00  -1.00   5.00  0.0000   1.0000   0.0000
-50.00   0.00   8.00  0.0000   0.9487  -0.3162

vnorm
4
-50.00   0.00   8.00  0.0000   0.9487  -0.3162
-50.00  -1.00   5.00  0.0000   1.0000   0.0000
-40.00  -1.00   5.00  0.0000   0.0000   1.0000
-40.00   0.00   9.00 -0.0353  -0.7067   0.7067

vnorm
3
-43.00  -1.00   5.00 -0.7071  -0.7071   0.0000
-40.00  -4.00   5.00 -1.0000   0.0000   0.0000
```

```
 -40.00  -1.00   5.00  0.0000  0.0000  1.0000

#/* TOP OF HULL */
Material
dark_grey
vnorm
4
 -70.00   3.00  -7.00  0.0000  1.0000  0.0000
 -70.00   3.00   7.00  0.0000  1.0000  0.0000
 -60.00   3.00   8.50  0.0000  1.0000  0.0000
 -60.00   3.00  -8.50  0.0000  1.0000  0.0000

vnorm
4
 -60.00   3.00  -8.50  0.0000  1.0000  0.0000
 -60.00   3.00   8.50  0.0000  1.0000  0.0000
 -50.00   3.00   9.50  0.0000  1.0000  0.0000
 -50.00   3.00  -9.50  0.0000  1.0000  0.0000

vnorm
4
 -50.00   3.00  -9.50  0.0000  1.0000  0.0000
 -50.00   3.00   9.50  0.0000  1.0000  0.0000
 -40.00   3.00  10.00 -1.0000  0.0000  0.0000
 -40.00   3.00 -10.00 -1.0000  0.0000  0.0000

snorm
4
 -40.00   3.00 -10.00
 -40.00   3.00  10.00
 -40.00   6.00  10.00
 -40.00   6.00 -10.00
 -1.0000  0.0000  0.0000

vnorm
4
 -40.00   6.00 -10.00  0.0000  1.0000  0.0000
 -40.00   6.00  10.00  0.0000  1.0000  0.0000
  42.00   6.00  10.00 -0.0499  0.9988  0.0000
  42.00   6.00 -10.00 -0.0499  0.9988  0.0000

vnorm
4
  42.00   6.00 -10.00 -0.0499  0.9988  0.0000
  42.00   6.00  10.00 -0.0499  0.9988  0.0000
 52.00   6.50   9.00  -0.0499  0.9988  0.0000
 52.00   6.50  -9.00  -0.0499  0.9988  0.0000

vnorm
4
 52.00   6.50  -9.00  -0.0499  0.9988  0.0000
 52.00   6.50   9.00  -0.0499  0.9988  0.0000
 62.00   7.00   7.00   0.1240 -0.9923  0.0000
 62.00   7.00  -7.00   0.1240 -0.9923  0.0000

vnorm
4
 62.00   7.00  -7.00   0.1240 -0.9923  0.0000
 62.00   7.00   7.00   0.1240 -0.9923  0.0000
 66.00   7.50   6.00  -0.0712  0.9975  0.0000
 66.00   7.50  -6.00  -0.0712  0.9975  0.0000

vnorm
4
 66.00   7.50  -6.00  -0.0712  0.9975  0.0000
 66.00   7.50   6.00  -0.0712  0.9975  0.0000
 73.00   8.00   4.00  -0.0995  0.9950  0.0000
 73.00   8.00  -4.00  -0.0995  0.9950  0.0000

vnorm
3
 73.00   8.00  -4.00  -0.0995  0.9950  0.0000
 73.00   8.00   4.00  -0.0995  0.9950  0.0000
 83.00   9.00   0.00  -0.0995  0.9950  0.0000

#/* REAR SUPER STRUCTURE */
#/* BASE OF SUPER STRUCTURE */
Material
Haze_grey
vnorm
4
 -21.75   9.00   5.30  -0.9701  0.2425  0.0000
 -22.50   6.00   5.30  -0.9701  0.2425  0.0000
   0.00   6.00   5.30   0.9738  0.2272  0.0000
  -0.70   9.00   5.30   0.9738  0.2272  0.0000

vnorm
4
   0.00   6.00   5.30   0.9738  0.2272  0.0000
   0.00   6.00  -5.30   0.0000  0.0000 -1.0000
 -0.70   9.00  -5.30  0.0000  0.0000 -1.0000
 -0.70   9.00   5.30  0.9738  0.2272  0.0000

vnorm
4
  0.00   6.00  -5.30  0.0000  0.0000 -1.0000
 -2.00   6.00  -5.30  0.8321  0.0000 -0.5547
 -2.00   9.00  -5.30  0.8321  0.0000 -0.5547
 -0.70   9.00  -5.30  0.0000  0.0000 -1.0000

vnorm
4
 -2.00   6.00  -5.30  0.8321  0.0000 -0.5547
 -4.00   6.00  -8.30  0.0000  0.0000 -1.0000
 -4.00   9.00  -8.30  0.0000  0.0000 -1.0000
 -2.00   9.00  -5.30  0.8321  0.0000 -0.5547

vnorm
4
 -4.00   6.00  -8.30  0.0000  0.0000 -1.0000
-22.50   6.00  -8.30 -0.9701  0.2425  0.0000
-21.75   9.00  -8.30 -0.9701  0.2425  0.0000
 -4.00   9.00  -8.30  0.0000  0.0000 -1.0000

vnorm
4
-22.50   6.00  -8.30 -0.9701  0.2425  0.0000
-22.50   6.00  -2.30 -0.9701  0.2425  0.0000
-21.75   9.00  -2.30 -0.9701  0.2425  0.0000
-21.75   9.00  -8.30 -0.9701  0.2425  0.0000

vnorm
4
-22.50   6.00   2.30 -0.9701  0.2425  0.0000
-22.50   6.00   5.30 -0.9701  0.2425  0.0000
-21.75   9.00   5.30 -0.9701  0.2425  0.0000
-21.75   9.00   2.30 -0.9701  0.2425  0.0000

#/* REAR PANEL OF REAR SS STACK */
snorm
4
-23.00   6.00  -1.88
-23.00   6.00   1.88
-23.00  11.30   1.65
-23.00  11.30  -1.65
```

```
-1.0000  0.0000  0.0000

#/* TOPS OF REAR SS STACK */
snorm
4
 -23.00  11.30  -1.65
 -23.00  11.30   1.65
 -17.50  11.30   1.88
 -17.50  11.30  -1.88
  0.0000  1.0000  0.0000

snorm
4
 -17.50  11.30  -1.88
 -17.50  11.30   1.88
 -17.50  14.00   1.65
 -17.50  14.00  -1.65
 -1.0000  0.0000  0.0000

snorm
4
 -17.50  14.00  -1.65
 -17.50  14.00   1.65
 -13.50  14.00   1.88
 -13.50  14.00  -1.88
  0.0000  1.0000  0.0000

snorm
4
 -13.50  14.00  -1.88
 -13.50  14.00   1.88
 -13.50  16.60   1.65
 -13.50  16.60  -1.65
 -1.0000  0.0000  0.0000

snorm
4
 -13.50  16.60  -1.65
 -13.50  16.60   1.65
  -9.50  16.60   1.88
  -9.50  16.60  -1.88
  0.0000  1.0000  0.0000

snorm
4
```

```
 -9.50  16.60  -1.88
 -9.50  16.60   1.88
 -9.50  18.30   1.65
 -9.50  18.30  -1.65
 1.0000  0.0000  0.0000

snorm
4
 -9.50  18.30  -1.65
 -9.50  18.30   1.65
 -2.00  18.30   2.50
 -2.00  18.30  -2.50
 0.0000  1.0000  0.0000
```

LIST OF REFERENCES

1. Adams, Rodney M. and Zyda, Michael J., *A Software Architecture for a Commander's Display System*, Master's Thesis, Naval Postgraduate School, Monterey, California, April 1987.
2. Silicon Graphics, Inc., *IRIS User's Guide*, Version 3.0, v. 1, Mountainview, California, 1987.
3. Hearn, Donald and Baker, M. Pauline, *Computer Graphics*, pp. 260-294, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1986.
4. Zyda, Michael J., Wu, C. Thomas, and Falby, John S., *A Data Structure for a Multi-Illumation Model Renderer*, Naval Postgraduate School, Monterey, California , December 1986.
5. Rogers, David F., *Procedural Elements for Computer Graphics*, pp. 309-345, McGraw-Hill, San Francisco, 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center	2
	Cameron Station	
	Alexandria, Virginia 22304-6145	
2.	Library, Code 0142	2
	Naval Postgraduate School	
	Monterey, California 93943-5002	
3.	Director, Information Systems (OP-945)	1
	Chief of Naval Operations	
	Navy Department	
	Washington, D.C. 20350-2000	
4.	Superintendent, Naval Postgraduate School	1
	Computer Technology	
	Monterey, California 93943-5000	
5.	Curriculum Officer, Code 37	1
	Computer Technology	
	Naval Postgraduate School	
	Monterey, California 93943-5000	
6.	Professor Michael J. Zyda, Code 52Zk	2
	Computer Science Department	
	Naval Postgraduate School	
	Monterey, California 93943-5000	
7.	Professor C. Thomas Wu	1
	Computer Science Department	
	Naval Postgraduate School	
	Monterey, California 93943-5000	
8.	LT Milton D. Abner	2
	720 Pinelake Dr.	
	Virginia Beach, Virginia 23462	